Programming Paradigms
CISC-280 Sample

Midterm II

NAME:

There are 100 points on 7 pages.

10 points

What could occur in the place of the “**X**” in the following Scheme evaluations?

```
(car (X (cdr '(a (b c) d)))) ⟶ B
```

```
(car (cdr (X (cdr '(1 (5 7) 8))))) ⟶ 7
```

```
(X '(whiskey vodka) '(lager cider)) ⟶ ((whiskey vodka) lager cider)
```

```
(X '((bill 10) (jill 14)) '((tracy 12) (jack 9)))
⟶ ((bill 10) (jill 14) (tracy 12) (jack 9))
```

```
(X (= 13 0) (/ 26 13)) ⟶ #f ;hint think “Special Forms”
```

6 points

Assume that the procedure `(enumerate-interval a b)` returns a list of integers starting at `a` and ending at `b`. Assume `(prime? x)` is a predicate that tests if `x` is a prime number. Assume `(square x)` returns x^2. Now, using the sequence operators `map`, `filter`, and `accumulate`, define a procedure named `foo` that finds the sum of the squares of all the primes from 1 to n.

10 points

The procedure square-list takes a list of numbers as arguments and returns a list of the squares of those numbers.

```
(square-list '(1 2 3 4)) ⟶ (1 4 9 16)
```

Here are two different definitions of square-list. Complete both of them by filling in the missing expressions:

```
(define (square-list items)
  (if (null? items)
      nil
      (cons ⟨ ?? ⟩  ⟨ ?? ⟩)))
```

```
(define (square-list items)
  (map ⟨ ?? ⟩ ⟨ ?? ⟩))
```

7 points

The procedure `scale-tree` scaled every element of a tree by some factor. It was defined as:

```
(define (scale-tree tree factor)
  (map (lambda (sub-tree)
         (if (pair? sub-tree)
             (scale-tree sub-tree factor)
             (* sub-tree factor)))
       tree))
```

Using that code as a guide, create a new procedure `tree-map`, that takes a procedure and a tree as arguments, and outputs a tree where every element has been replaced by the value of calling the procedure on the element. For example, we could now write `square-tree` as:

```
(define (square-tree tree) (tree-map square tree))
(square-tree '(1 (2 (3 4) 5) (6 7)) ⟶ (1 (4 (9 16) 25) (36 49))
```

```
(define (tree-map proc tree)
```

8 points

Define a procedure `make-list`, which takes a non-negative integer n and an object and returns a new list, of length n, where each element is the object.

```
(make-list 7 '()) ⟶ (() () () () () () ())
```

5 points

Define a procedure `remove` that removes all occurances of its first argument from the second argument (a list). Use `equal?` as a test.

```
(remove 'dog '(the brown dog bit the small dog))
     ⟶ (the brown bit the small)
```

3 points

Show the set [3 5 7 13 9 10] as a balanced binary tree (Draw the picture).

3 points

Assuming that we represent/implement a node in a balanced binary tree in Scheme as a list of (node-value left-subtree right-subtree), write down the Scheme representation of the tree you drew above. It would be helpful if you indented it nicely.

4 points

What advantages does the balanced binary tree set representation have over the ordered list representation? What advantages does the ordered list representation have over the balanced binary tree representation?

12 points

Define a generic predicate `=zero?` that tests if its argument is zero. Define and install data-directed implementations for rational numbers (type `'rational`; selectors `numerator` and `denomenator`), and complex numbers type `'complex`; selectors `real-part`, `imag-part`, `magnitude`, and `angle`). Don't forget the external interface.

12 points

Draw the environment diagram (all frames and user-defined procedure definitions) that results from executing the following three lines of Scheme code:

```
(define (damp f) (lambda (x) (/  (+ x (f x))   2)))
(define damped-sqrt (damp sqrt))
(damped-sqrt 4)
```

10 points

Assume I have already defined the procedure `symbol-append` which takes two symbols and creates a single symbol with the two stuck together like this:

```
(symbol-append 'apple 35) ⟶ apple35
```

A useful thing in writing large simulations is to have a unique name for every object in the simulation. To do this, we need to generate a unique symbol for the name. Define a procedure `gensym` that takes one argument `name`, and that generates a procedure that creates a new, ***unique*** symbol beginning with `name` each time it is called. Hint: use a counter and `set!`. Example:

```
(define gentrains (gensym 'train))
(gentrains) ⟶ train1
(gentrains) ⟶ train2
(gentrains) ⟶ train3
```

10 points

```
(define (make-withdraw balance)
  (lambda (amount)
    (if (>= balance amount)
        (begin (set! balance (- balance amount))
               balance)
        ``Insufficient funds'')))
```

Modify make-withdraw so that is creates a password-protected account. Make-withdraw will thus take two args, the initial balance and the real password. The resulting function should also take TWO arguments, the amount to withdraw and a password. It should only allow the withdrawal if the passwords match. Otherwise it should return “Incorrect password”.